Improving Performance of Clusters using Load Balancing Algorithms

Thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Argha Sen

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela, Odisha, 769 008, India

May 2011

Improving Performance of Clusters using Load Balancing Algorithms

Thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Argha Sen

(Roll- 107CS004)

Under the guidance of

Prof.Pabitra Mohan Khilar

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela, Odisha, 769 008, India

May 2011

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

Certificate

This is to certify that the work in the thesis entitled ***Improving Performance of Clusters using Dynamic Load Balancing Algorithms*** submitted by ***Argha Sen*** is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering in the department of Computer Science and Engineering, National Institute of Technology Rourkela. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Place: NIT Rourkela
Date: 9 May 2011

Dr. Pabitra Mohan Khilar
Assistant Professor
Dept. of Computer Science and Engineering
National Institute of Technology, Rourkela
Odisha-769 008

Acknowledgment

It is a pleasure to thank those who made this thesis possible. At the outset, I would like to express my sincere thanks to Prof. Pabitra Mohan Khilar for his advice during my thesis work. As my supervisor, he has constantly encouraged me to remain focused on achieving my goal. His observations and comments helped me to establish the overall direction of the research and to move forward with investigation in depth.

I am very much indebted to Prof. Ashok Kumar Turuk, Head-CSE, for his continuous encouragement and support. I am also thankful to all the professors of the department for their support.

I am grateful to Mr.Arijit Mukherjee,M.Tech scholar,NIT Rourkela who has made available his support in a number of ways. I am also thankful to my all friends who have provided me with kind words, new ideas, useful criticism, or their invaluable time.

Last, but not the least, I would like to dedicate this thesis to my family, for their love, patience, and understanding.

Argha Sen

Abstract

A Cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected stand alone computers working together as a single integrated computing resource generally having a single system image **(SSI)**, that is the users generally view the clusters as a single system. A Cluster can be used in scientific applications that need supercomputing power or in various other areas like databases, multimedia, web services, etc. In addition the users can access any node of the cluster. Load balancing tries to balance the total cluster system load by transferring or starting processes on the more lightly loaded nodes in preference to heavily loaded nodes. .Many load balancing algorithms have been proposed such as round-robin (RR) scheduling, weighted round-robin (WRR) scheduling, least-connection (LC) scheduling and WLC (Weighted Least Connection) scheduling. Comparative studies on their performance have thrown light on their merits and drawbacks with regards to various parameters. This thesis deals with the study of various load balancing algorithms and an attempt at producing an algorithm that can improve the cluster performance.

The proposed algorithm can be viewed a two stage process to balance the load in Heterogeneous Clusters though it works perfectly for homogeneous clusters as well. In the first stage the nodes are analyzed to produce their performance factor and in the second stage the load balancing is done. The algorithm is validated with simulations done in MATLAB and comparing the results with existing algorithms for load balancing in clusters

Contents

List of Figures

Chapter 1

Introduction

An overview of Cluster Computing

Load Balancing in Cluster Computing

Key Goals of Load Balancing Algorithms

Classification of Load Balancing Algorithms

Problem Formulation

Thesis Organization

Chapter 1

Introduction

Cluster Computing has come a long way since the early beginnings in the 1960s with the advent of high performance microprocessors and high speed networks. This has gained further momentum with the development of standard tools for high performance distributed computing. Clusters give us the advantage of using cheap PCs and workstations over a network that provides us a cost effective form of parallel computing. The increasing need of computing power, the prohibitive cost of supercomputers and their low accessibility have all led to the research in clusters that are providing us services similar to supercomputers at a low cost. In fact there are a very good number of clusters in the TOP 500 list of computing resources.

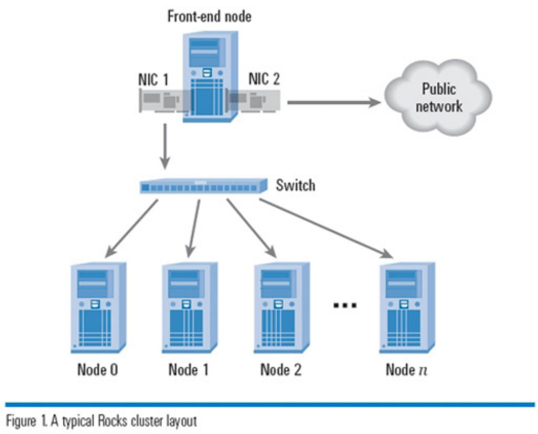

Figure 1.1: Cluster Layout

1.1 An overview of Cluster Computing

A computer cluster is a group of linked computers, working together closely thus in many respects forming a single computer [1]. The components of a cluster are commonly, but not always, connected to each other through fast local area networks [2]. Clusters are usually deployed to improve performance and/or availability over that of a single computer, while typically being much more cost-effective than single computers of comparable speed or availability [3].Scalable Parallel Computers require good performance, low latency and high bandwidth communications, scalable network bandwidth, and fast access to files. Clusters can meet these requirements by using resources of its nodes. A File system supporting parallel I/O can be built using these disks associated with each node instead of the conventional RAID (Redundant Array of Inexpensive Disks).

The cluster nodes can work collectively, as an integrated computing resource, or they can operate as individual computers. The cluster middleware is responsible for offering an illusion of a unified system image **(SSI)** and availability of the collection of independent but interconnected computers. Clusters can be used for the execution of both sequential as well as parallel applications.

Resource Management and Scheduling**(RMS)**is the cat of distributing applications among computers to maximize their throughput. It also enables the effictive and efficient use of the resources available. Thus RMS envioronments provide middleware services to users that enables them to use heterogeneous environments of workstations efficiently and easily. The services provided by RMS include [1]:

- Process Migration.
- Checkpointing.
- Scavenging Idle Cycles.
- Fault Tolerance.

- Minimization of Impact on users.
- Load Balancing.
- Multiple Application Queues.

Each of the RMS services is a vast topic in itself with algorithms of their own and this thesis deals with the topic of Load Balancing.

Load Balancing refers to the manner in which the jobs are distributed among all the computational platforms available in a particular organization. This enables the usage of all the resources, rather than the few which is known to the users [1].

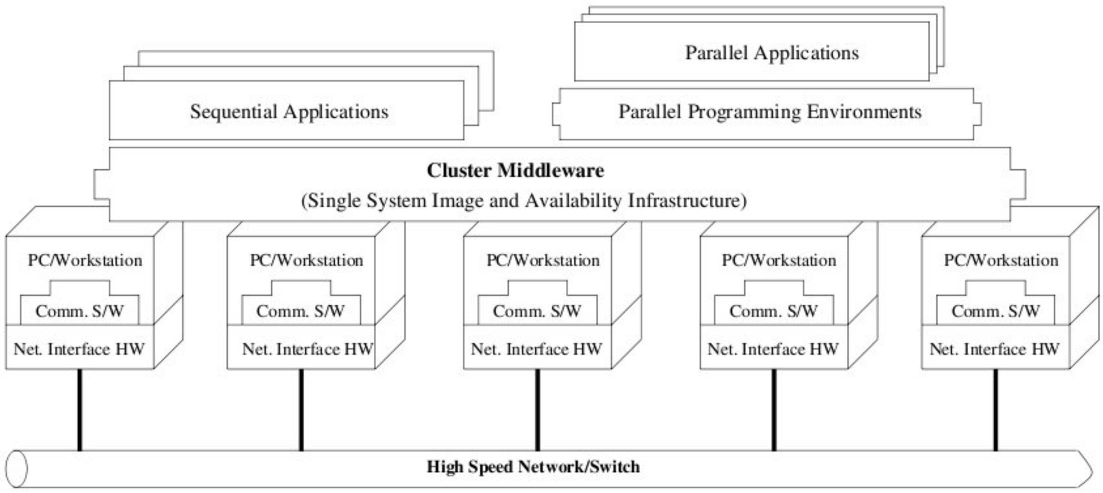

Figure 1.2: Cluster Computing Architecture adapted from [1]

1.2 Load Balancing in Cluster Computing

Load Balancing is a technique to distribute workload evenly across two or more computers, network links, CPUs, hard drives, or other resources, in order to get optimal resource utilization, maximize throughput, minimize response time, and avoid overload. It can therefore avoid situations where some nodes are heavily loaded while others are idle or doing little work.Using several hosts to provide services together provides high performance-price ratio, high reliability and scalability, whose target is to reasonably assign the task to each server of cluster and make each server work more availably, and thus it raises the service quality of the whole server system [4]. Process Migration is an important strategy of the load balancing where it may be beneficial to move processes from overloaded system to lightly loaded ones.

An Effective Load Balancing algorithm needs to take care of various factors of performance evaluation which includes parameters such as load estimation, load levels, comparison, performance indices, system stability, communication latency, communication data volume, remote node selection etc.

1.3 Key Goals of Load Balancing Algorithms

1. Its Primary aim is to achieve an overall improvement in system performance at a reasonable cost. An Algorithm that balances the load perfectly but uses a lot of computation or communication is of not much use.

2. To treat all jobs in the system equally regardless of their origin. However there may be priorities attached to the nodes or jobs which has to be taken care of by the algorithm.

3. Scalability of the algorithm is very important as the distributed system in which it is implemented may change in size or topology. Hence the algorithm must be flexible enough to allow such changes to be handled easily.

4. It needs to have a degree of fault tolerance i.e. it must be able to recover in case some nodes go down or give faulty results.

5. It has to maintain system stability.

1.4 Classification of Load Balancing Algorithms

Load balancing Algorithms are classfied into the following subtypes:

On the basis of Initiation:

Sender Initiated: In this type of algorithm the sender sends request messages till it finds a receiver that can accept the load.

Receiver Initiated: In this type of description algorithms the receiver sends request messages till it finds a sender that can get the load.

Symmetric: It is a combination of sender and receiver initiated type algorithms.

On the knowledge of Nodes:

Static: In this method the performance of the processors is determined at the beginning of execution. Then depending upon their performance the work load is distributed in the start by the master processor. The slave processors calculate their allocated work and submit their result to the master. A task is always executed on the processor to which it is assigned that is static load balancing methods are non-preemptive. The goal of static load balancing method is to reduce the overall execution time of a concurrent program while minimizing the communication delays. A general disadvantage of all static schemes is that the final selection of a host for process allocation is made when the process is created and cannot be changed during process execution to make changes in the system load [5].

Dynamic : In Dynamic load balancing approach, the decision on load balancing is based on the distributed system. Tasks are allowed to move from an overloaded system to under loaded system to receive faster service. Dynamic approach is better as it considers the state of the system in making the load balanced. It is of 2 types:

1. Distributed: In this system the Load Balancing Algorithm is executed by all the nodes in the system and load balancing responsibility is shared by all the nodes. Distributed Dynamic Load Balancing Algorithms generally tend to produce more message passing as due to the distributed approach each node need to communicate with each other to make the decisions.
2. Non-Distributed: In this system the nodes are not responsible for load balancing. Instead a load balancing master is chosen who does the load balancing for the entire system.

Based on interaction between nodes:

Cooperative: In Cooperative systems, nodes work closely to achieve a global objective i.e. to improve the system's overall response time.

Non-Cooperative: In Non-Cooperative systems, nodes work independent of each other in order to obtain a local goal i.e to improve the individual system's response time.

1.5 Problem Formulation

The main goal is to design and implement Load Balancing Algorithm that will operate in clusters and try to improve its performance. The algorithm will consider the load states of all the nodes before deciding on the node to run the next process. In order to design and implement this following have to achieve:

- Study of the formal definition of Load Balancing and the models of Load.
- Study of the existing algorithms on Load Balancing to develop a base for the development of proposed Load Balancing Algorithm.
- Study of Message Passing Interface(MPI) for implementation of parallel modules.
- Implementation of the various algorithms for comparison and analysis

1.6 Thesis Organization

The rest of the thesis is organized as follows:
In Chapter 2, an overview of Load Balancing Algorithms used in clusters are provided. It also discusses various literatures surveys related to work. It also gives the overview related to the message passing interface.

In Chapter 3, we propose a mechanism to improve the cluster performance using load balancing.It also provides an overview of efficient comunication model in clusters. It also discusses the simulations and results of our proposed scheme.

Finally in Chapter 4, we discuss about the conclusions and scope for the future work.

Chapter 2

Literature Survey

Chapter 2

Literature Survey

2.1 Study Of Existing Load Balancing Algorithms

In this section we see a few of the existing Load Balancing Algorithms. A total of 8 algorithms are described below four each from the static and dynamic types of classification.

2.1.1 Round Robin

In the round robin algorithm [6] processes are divided evenly between all processors. Each new process is assigned to new processor in round robin order. The process allocation order is maintained on each processor locally independent of allocations from remote processors [7]. With equal workload round robin algorithm is expected to work well [5]. However when the jobs are of unequal processing time this algorithm suffers as the some nodes can become severely loaded while others remain idle. Round Robin is generally used in web servers where generally HTTP requests are of similar nature and thereby be distributed equally.

2.1.2 Randomized Algorithms

The randomized algorithm [8] is an static algorithm applies a probabilistic approach as opposed to the deterministic approach followed in Round Robin Scheduling. In this method a process can be handled by a particular node n with a probability p. This also works very well in case each process loads is equal but fails if loads are of various computational complexities. It is used in alteration to round robin algorithm when there are large numbers of nodes as maintaining the queue of nodes for round robin become an overhead.

2.1.3 Central Manager Algorithm

The algorithm [9] has a central processor which selects the host for new process. The minimally loaded processor depending on the overall load is selected when process is created. Load manager selects hosts for new processes so that the processor load confirms to same level as much as possible. The central load manager makes the load balancing judgment from the on hand information on the system load state. This information is updated by remote processors, which send a message each time the load on them changes [5]. The load manager makes load balancing decisions based on the system load information, allowing the best decision when of the process created. High degree of inter-process communication could make the bottleneck state. This algorithm is expected to perform better than the parallel applications, especially when dynamic activities are created by different hosts [5].

2.1.4 Threshold Algorithm

According to this algorithm, the processes are assigned immediately upon creation to hosts. Hosts for new processes are selected locally without sending remote messages. Each processor keeps a private copy of the system's load. The load of a processor can characterize by one of the three levels: underloaded, medium and overloaded. Two threshold parameters tunder and tupper can be used to describe these levels.

Under loaded - load $<$ tunder

Medium - tunder $\leq$ load $\leq$ tupper

Overloaded - load $>$tupper

Initially, all the processors are considered to be underloaded. When the load state of a processor exceeds a load level limit, and then it sends messages regarding the new load state to all remote processors, regularly updating them as to the actual load state of the entire system. If the local state is not overloaded then the process is allocated locally. Otherwise, a remote under loaded processor is selected, and if no such host exists, the process is also allocated locally. Thresholds algorithm have low inter process communication and a large number of local process allocations. The later decreases the overhead of remote process allocations and the overhead of remote memory accesses, which leads to improvement in performance. A disadvantage of the algorithm is that all processes are allocated locally when all remote processors are overloaded. A load on one overloaded processor can be much higher than another overloaded processors, causing significant disturbance in load balancing, and increasing the execution time of an application [5].

2.1.5 Central Queue Algorithm

Central Queue Algorithm [10] works on the principle of dynamic distribution. It stores new activities and unfulfilled requests as a cyclic FIFO queue on the main host. Each new activity arriving at the queue manager is inserted into the queue. Then, whenever a request for an activity is received by the queue manager, it removes the first activity from the queue and sends it to the requester. If there are no ready activities in the queue, the request is buffered, until a new activity is available. If a new activity arrives at the queue manager while there are unanswered requests in the queue, the first such request is removed from the queue and the new activity is assigned to it. When a processor load falls under the threshold, the local load manager sends a request for a new activity to the central load manager. The central load manager answers the request immediately if a ready activity is found in the process-request queue, or queues the request until a new activity arrives [5].

2.1.6 Local Queue Algorithm

Main feature of this algorithm [11] is dynamic process migration support. The basic idea of the local queue algorithm is static allocation of all new processes with process migration initiated by a host when its load falls under threshold limit, is a user-defined parameter of the algorithm. The parameter defines the minimal number of ready processes the load manager attempts to provide on each processor. Initially, new processes created on the main host are allocated on all under loaded hosts. The number of parallel activities created by the first parallel construct on the main host is usually sufficient for allocation on all remote hosts. From then on, all the processes created on the main host and all other hosts are allocated locally [5].

When the host gets under loaded, the local load manager attempts to get several processes from remote hosts. It randomly sends requests with the number of local ready processes to remote load managers. When a load manager receives such a request, it compares the local number of ready processes with the received number. If the former is greater than the latter, then some of the running processes are transferred to the requester and an affirmative confirmation with the number of processes transferred is returned [5].

2.1.7 Least Connection Algorithm

The least-connection scheduling algorithm directs network connections to the server with the least number of established connections. This is one of the dynamic scheduling algorithms; because it needs to count the number of connections for each server dynamically to estimate its load. The load balancer records the connection number of each server, increases the connection number of a server when a new connection is dispatched to it, and decrease the connection number of a server when a connection finishes or timeouts. The formal procedure of round-robin scheduling is as follows [2]:

Supposing that there is a server set S = {S0, S1, ..., Sn-1},
W(Si) is the weight of server Si,
C(Si) is the current connection number of server Si,
for (m = 0; m < n; m++) {
if (W(Sm) > 0) {

for (i = m+1; i ¡ n; i++){

if (W(Si) ≤ 0)

continue;

if (C(Si) < C(Sm))

m = i; }

return Sm;}

return NULL;

}

2.1.8 Weighed Least Connection Algorithm

The weighted least-connection scheduling is a superset of the least-connection scheduling, in which you can assign a performance weight to each real server. The servers with a higher weight value will receive a larger percentage of active connections at any one time [12]. The default server weight is one, and the IPVS Administrator or monitoring program can assign any weight to real server. In the weighted least-connections scheduling, new network connection is assigned to a server which has the least ratio of the current active connection number to its weight. The formal procedure of is as follows [2]:

Supposing there is a server set S = {S0, S1, ..., Sn-1},

W(Si) is the weight of server Si;

C(Si) is the current connection number of server Si;

CSUM = Σ C(Si) (i=0, 1, .. , n-1) is the sum of current connection numbers;

for (m = 0; m < n; m++) {

if (W(Sm) > 0) {

for (i = m+1; i < n; i++) {

*if (C(Sm)*W(Si) > C(Si)*W(Sm))*

m = i;

}

return Sm;

}

```
    }
    return NULL;
}
```

2.2 Efficient Load Balancing Algorithms based on WLC

1. **Algorithm By Chunkyon Young and Ilyong Chung** [13]

   This Algorithm is based on the fact that WLC algorithm does not use the accurate load status for servers. The load unbalancing markers used by them are

   - CPU Load
   - Load of Memory
   - Load of Network

   Using these three they provide a load measuring algorithm that can determine if the server load is high. This algorithm is loaded on all servers and is called by a Central Monitor. The load measuring algorithm can be explained by the following pseudo code [13]:

```
void Load_Status_col()
{
          if(check_waiting_process_exists()==y)
                    set_order=4;
          else if(free_memory<=init_average)
                    set_order=3;
          else if(collision_ratio>=y$%$)
                    set_order=2;
          else
                    set_order=1;
}
```

   The parameters ‘y’and ‘init_average’are set in accordance to the server capabilities as well as network characteristics.The load balancing algorithm need the “order”to be set so as to balance the load. The load is balanced by the following algorithm [13]:

```
void Load_Dis_Start()
{
if(load_order_set!=1){
        while(R!=n){
                R=R+1;
        if(load_order_set==2)flag=y;
        else if(load_order_set==3)flag=y;
        else if(load_order_set==4)flag=y;
        else{
                sleep(k);
                load_dis_start_deamon();}
        if(flag==y){
        if(check_only_one())
        client_connect= True;
        else{
                server=get_minimum_load_server()
                client_connect= True;}
}}}}
```

2. **Algorithm by Paul Werstien,Hailing Situ,Zhiyi Huang** [14]
This Algorithm assumes a homogeneous environment for clusters running the Linux Operating system and a file sharing system on NFS. The load estimation is done on CPU utilization, memory utilization, network traffic etc.However, it considers the average usage during a time interval ‘t’. The load metric is calculated as per the following equation:

$$l_{n\{par\}} = \frac{p_1 + p_2 + \cdots + p_t}{t}$$

Here,$l_{n\{par\}}$= load average metric for a particular parameter ‘par’
$p_1 \ldots p_t$= load values in that particular one second interval
t=no. of time intervals

After calculation of load metric on each node it is again averaged over the'n'nodes of the cluster.

$$l_{avg\{par\}} = \frac{l_1 + l_2 + \cdots + l_n}{n}$$

this is used to calculate the threshold values

$$t_H = H * l_{avg}, t_L = L * l_{avg}$$

These threshold values are for all parameters and are based on the individual parameters.
Based on the threshhold values nodes are classified as idle,low,normal and high.These classification are made by weighing the parameters.Finally the decision to run the process is taken by the following algorithm [14].

```
if(localhost==idle)
        run localhost
else if(idleNodesAvailable())
         run idlenode
else if(localhost==high&&lowLoadedNodesAvailable())
        run lowLoadedNode
else
         run localhost
end if
```

2.3 Observations from the study of Various Algorithms

1. If more than one parameter is used for load estimation the estimation is more reliable than cases where only one parameter is used.

2. If individual nodes calculate their load status, the network load of running the algorithm is reduced.

3. Threshold values should be 2 or more in order to allow smooth transition from under-loaded to overloaded conditions. If only one threshold is used the fluctuations are high and the performance may degrade.

4. The condition that many nodes simultaneously select a certain node for running a process must be avoided.

5. In case of heterogeneous nodes, weights can be assigned so as to ensure that the differences in amount of resources can be accounted for.

2.4 Heterogeneous clusters and their effective use

Clusters are tried to be kept as homogeneous as possible, to maintain the Single System Image (SSI) Thus under the usual circumstances, a heterogeneous cluster seems undesirable. However, it is sometimes deliberately needed to include heterogeneous nodes to take advantage of the performance of certain architectures and other advanced components.

Moreover, the improved hardware in newer machines must be taken advantage of else there is no use investing in such hardware. But these must be used in conjunction with the other available machines. Thus, the concept of heterogeneous clusters in the real world is indeed a very important one.

2.4.1 Defining Heterogeneous clusters for the current study

For the present study, the term "Heterogeneous Clusters have been defined as follows:- The individual nodes may have different computing power as CPU speed may vary, different amount of Memory available as well as different vendor specific architectures.

On the OS front, it has been assumed that all the nodes are running a Linux based OS for the hardware related information is stored in them in a similar structured file system. The heterogeneity if extended to Windows based OS will increase the complexity of the algorithm because of the difference of information structure on the Linux and Windows OS.

The seemingly constraint choice OS is not very significant as the Cluster middleware are on the GNU license, which effectively port the SSI image to Windows OS. Also the use of Virtual Machines in the system will enable the Linux Kernel to run on top of the Windows based OS.

2.4.2 Dealing with Heterogeneity in Clusters

In order to deal with the heterogeneity in the clusters, we have a four step paradigm as explained below [11]:-

1. **State Measurement :** This stage lies with the load estimation.
2. **Information Exchange :**It is a stage where the communication overheads need to be considered in order to ensure which method of exchanging load states would be beneficial to the algorithm.
3. **Initiation Rule:** The load balancing algorithm should be initiated only when the benefits of load balancing algorithm outweigh the cost of running the algorithm.
4. **Load Balancing Operation:** This step is further subdivided into 3 more steps-the location, distribution and selection rules. They deal with the way the algorithm distributes the load and on the nodes the load goes.

2.5 Summary

In the above sections of the chapter we have studied the various load balancing algorithms in both homogeneous and heterogeneous clusters.We have also drawn various observations from the algorithms which have formed the very foundation of our algorithm. We have also defined what the scope of the heterogeneous clusters with regard to the current study and have studied the way of dealing with them .

Chapter 3

Our Work

Chapter 3

Our Work

3.1 Introduction

In this chapter we present a couple of algorithms, the first one is a simple algorithm to homogenize the nodes of a heterogeneous system. This enables us to use the load balancing algorithms for homogeneous systems in heterogeneous systems. This also automates the process of calculating the weights of different servers used by the popular algorithms like WLC.

The second algorithm forms the main part of this thesis and is an algorithm for load balancing in clusters. The algorithm has been developed for the homogeneous family of clusters and may be extended to the heterogeneous family of clusters using the first algorithm. Thus in case of heterogeneous clusters we can think the algorithms to be 2 stages to achieve the goal of load balancing.

3.2 Algorithm to homogenize the nodes

In this section an algorithm has been developed which determines the CPU characteristics of a node and facilitates the assignments of weights to all the nodes. As a first step we have to determine the individual node characteristics by the following Algorithm.

3.2.1 Determine CPU characteristics:

The following Pseudo-code reads the contents of the **/proc/cpuinfo** file and gives us the CPU charcteristics in a Linux based system.

```
void get_cpu_params ()
{
  openfile ("/proc/cpuinfo", "r");
  bytes_read = fread (buffer, 1, sizeof (buffer), fp);
  if (bytes_read == 0 || bytes_read == sizeof (buffer))
    return ;
  get_param_val(buffer,"model name"));
  get_param_val(buffer,"cpu MHz");
  get_param_val(buffer,"cache size");
  get_param_val(buffer,"bogomips");
}
```

3.2.2 Communicating data between the nodes

We assume a Tree-Based Communication pattern in which we send the CPU characteristics data in stages.In this type of communication at a particular stage n we send data n new nodes. Here,we assume a particular node to be node 0(root),and the data is then sent in stages.In stage 1,node 0 sends to node 1. In stage 2,node 0 sends to node 2 and node 1 sends to node 3.Thus in this process if the total no of nodes is p we need $log_2 p$ stages [15].

Algorithm for the tree based communication pattern.

```
for(stage=first;stage<=last;stage++)
{
        if(I_Recieve(stage,my_rank,&source))
                Recieve(data,source);
        else if(I_send(stage,my_rank,&dest))
```

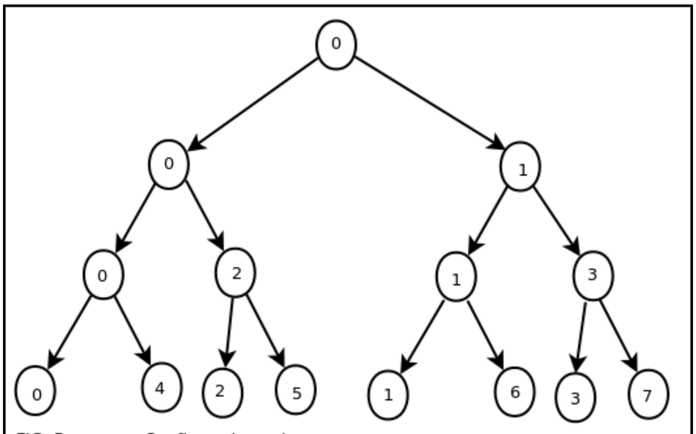

Figure 3.1: Processes configured as a tree.

```
        Send(data, dest);
}
```

The I_ Recieve function checks if during the current stage the current process recieves data.Similarly,the I_Send function checks if during the current stage the process sends data.
However the sending pattern is diffult to find out optimally without the knowledge of the topology. However there is a simple implementation algorithm which is applied to the problem [15].

if(2^{stage} < my_ rank < $2^{stage+1}$)
    Recieve (my_ rank - 2^{stage})
else if(my_ rank < 2^{stage})
    Send(my_ rank + 2^{stage})

3.3 Algorithm for Load Balancing in Clusters

This section presents an algorithm for the load balancing. This uses the results obtained from the pervious section in case it is used for heterogeneous systems.

3.3.1 Initiation Step-Leader Election and Performance Factor

In order to initiate and monitor the load balancing of the cluster we need to define a **Leader Election Algorithm**.This can be done by using any one of the various algorithms available for the same purpose.

However, instead of going for an algorithm as such, we use a heuristic here that the fastest processor as determined by the following formula can be selected as the leader :-

Performance facctor of a processor $i(\chi_{\mathbf{i}})=\frac{\sum_{p=1}^{n} w_p P_p}{\sum_{p=1}^{n} w_p}$

Here $\mathbf{w_p}$ is the value set on each parameter as per need for the cluster. w_p can be varied as per the specific requirements.

$\mathbf{P_p}$=Value of the parameter for that specific node.

Then the leader is elected as such by the formula:

$$\textbf{Leader} = Max(\chi_i)$$

The value of i for which χ_i is maximum is therefore taken as the leader in the system.

3.3.2 Obtaining Load Characteristics

The load characteristics need to be obtained from all the nodes as the next step of the algorithm. This can be done by taking characteristics of the profiling done by Linux operating system. For this the files of the “/proc ”directory of the OS is parsed. CPU load is obtained from the “/proc ”directory in the /proc/stat file. Memory load is calculated from /proc/meminfo.

The network load is obtained from the/proc/net/dev file. Here we only consider the active device and no arbitrary use of passive devices is considered .As most networks will only use one active device the estimation from this method is fairly accurate.

This gives us fairly good load characteristics of the node.

For a particular task we can get characteristics by a similar technique. We here parse the files by the /proc/<current_process_id >/status files. We obtain the process'CPU Load and memory usage.
To get the network load of a particular process we use a indirect technique as explained in [16].

3.3.3 Getting a comprehensive Load Estimate

The obtained load is the combined into a particular load value or **Load Index**. For this we need to assign weights to the CPU, Memory, and Network usage. The values of these weights will give us the flexibility of using the clusters for different types of jobs and only the weights need to be tweaked to estimate the load.

$$L_n = W_{CPU} * CPU_{load} + W_{MEM} * MEM_{load} + W_{NET} * NET_{load}$$

However if the loads are normalized into percentage usage values, it becomes easier to assign the loads. This process is simple as we have already obtained a copy of available resources in the previous steps.
the load index L_n will be sufficient for load estimation if the cluster is homogeneous.
In heterogeneous systems a simple change is needed as the load value will be scaled to account for heterogeneity.

$$L_n^{'} = \chi_n + L_n$$

where χ_n is the performance factor calculate in section 3.1.

3.3.4 Calculate Average Load

On obtaining load estimates, they can be broadcasted to all nodes or to the leader depending on the fact as to which type of load balancing algorithm we use in the following steps. We can reduce the number of messages sent by using the Leader elected in step 4.

The average load of the system can be calculated as

$$L_{avg} = \frac{\sum_{i=1}^{n} L_i}{n}$$

Once again in case of heterogeneous clusters we use the performance factor χ_n to counter it.

We have assumed two parameters of load determination Load_Lower_Limit and Load_Upper_Limit as folows:

Load_Lower_Limit=0.8

Load_Upper_Limit=1.2

This allows us a 40% range where load will be considered normal.

The values of Load_Lower_Limit and Load_Upper_Limit can be changed as per need to ensure load balancing. However, if the values are too close to each other too much fluctuations in load states will occur which would cause the following in load states would occur which would cause the following algorithm to slow down. If the values are too far apart, the load state would remain normal for a very long time which may in extreme cases lead to the non utilization of the load balancing algorithm. We have to check the load of every node and decide which node is in what state.

3.3.5 Determining Load States

For determining the load states we use $L_n^{'}$ instead of L_n. Thus, we will be defining the load limits as follows :

$L_n^{'}$ <Load_Lower_Limit $*L_{avg}$ = Under_loaded

$L_n^{'}$>Load_Lower_Limit $*L_{avg}$ & $L_n^{'}$<Load_Upper_Limit $*L_{avg}$ =Normal

$L_n^{'}$>Load_Upper_Limit $*L_{avg}$ =Over_Loaded

These states are fed into the final balancing algorithm.

3.3.6 Selection of Nodes for execution

The nodes of the cluster have been classified into 3 categories in section 3.2.5.We already have the load characteristics of each node from section 3.2.2.Using these we use the following algorithm to assign the process to node.

```
if(host==under_loaded)
        run=Host;
else if(host=normal||host=over_loaded)
{
        if(search_Under_loaded())
                run=assign_under_loaded();
        else if(search_normal())
        {
                if(host!=normal)
                        run=assign_normal();
                else if(host==normal)
                        run host;
        }
}
```

3.3.7 Update the load Estimate

Once a new process arrives the load values will have changed .So these are calculated and updated at all nodes. This means we need to update only one entry in the table of load estimates each time a process arrives at the node. So we need a broadcast message every time a process runs on the clusters. Again when the process completes execution the resources will have freed and the load states will again change. These will be updated by a broadcast message to all nodes. Thus the algorithm will incur an additional cost of 2 broadcast messages every process.

3.4 Simulation Of Algorithm and Comparison with other algorithms.

Assumptions:

1. Process Burst Times are in an exponential distribution ofλ =10
2. Process Start times are in Poisson distribution of $\lambda = 1$, with the axis shifted to 1 to ensure that no two process arrive at exactly the same interval of time.
3. Process memory load has been kept as a fixed and variable part. The fixed part has been set at 50 and the variable part is a uniform distribution of max 300.
4. Process CPU Load are in percentage figures and have been modeled as an exponential distribution of $\lambda = 10$.the assumption was based on the evaluation of CPU load over a period of time and reading their CPU utilization and curve fitting techniques.
5. Process Network Load has been simulated with a Gaussian distribution of μ =50 and σ =15. These values have been chosen because of the observation that most process has similar network usage in a cluster with a few extreme uses at both ends. The value of μ, σ gives a full 0-100 % utilization curve with a 1-σ range of 35-65 % of network utilization.

3.4.1 Simulation Results in Homogeneous Clusters

In Homogeneous clusters the algorithms simulated are Round Robin, Randomized and Central Manager algorithm along with the proposed algorithm. The results are presented in the following graphs:

Turnaround Times

The proposed algorithm gives results that are comparable in Turnaround times to all the three algorithms:

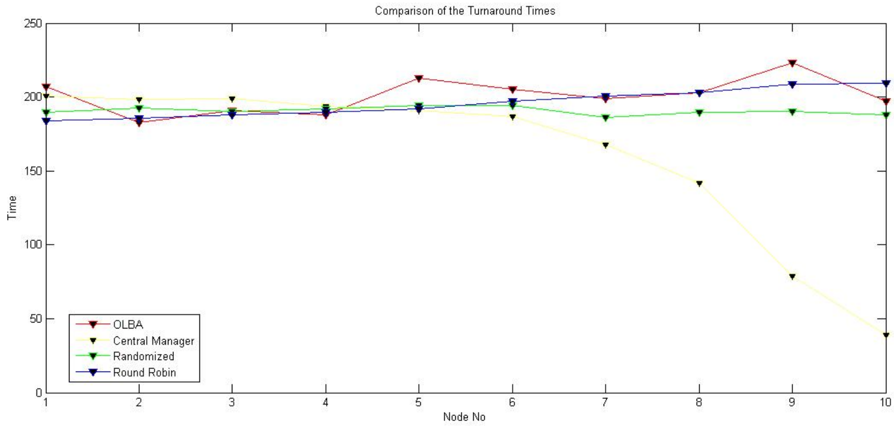

Figure 3.2: Turnaround Times Comparison in Homogeneous Algorithms.

Peak CPU Load

The proposed algorithm gives results that are better in CPU Usage to all the three algorithms:

Peak Network Load

The proposed algorithm gives results that are comparable in Network Usage to all the three algorithms:

Memory Load Profile of individual nodes.

The proposed algorithm gives the most balanced memory load profile among the four simulated algorithms.

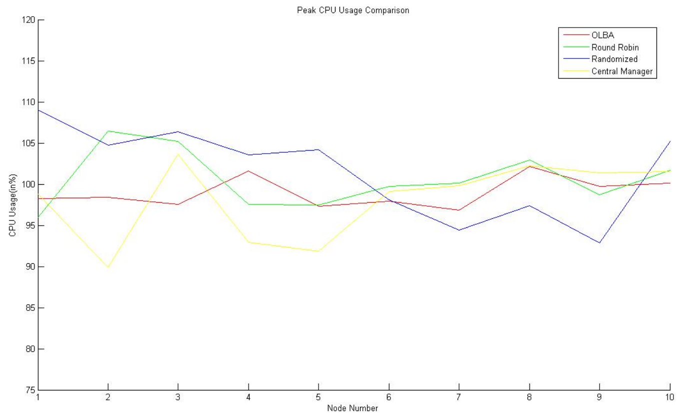

Figure 3.3: Peak CPU Load in Homogeneous Algorithms.

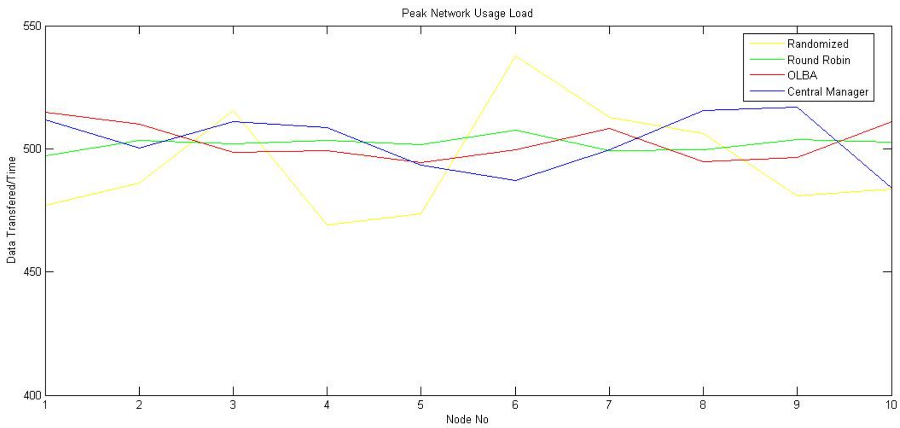

Figure 3.4: Peak Network Load in Homogeneous Algorithms.

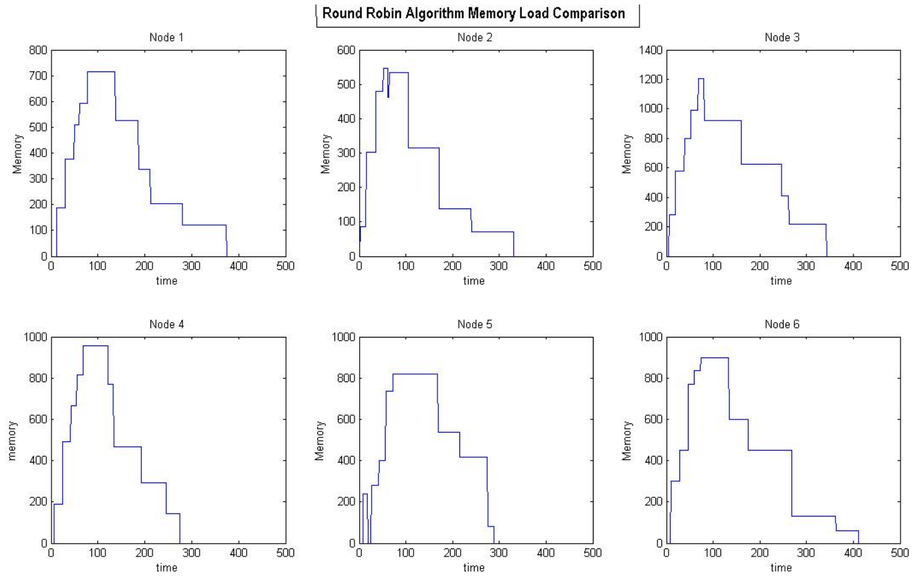

Figure 3.5: Round Robin Algorithm Memory Load Profile.

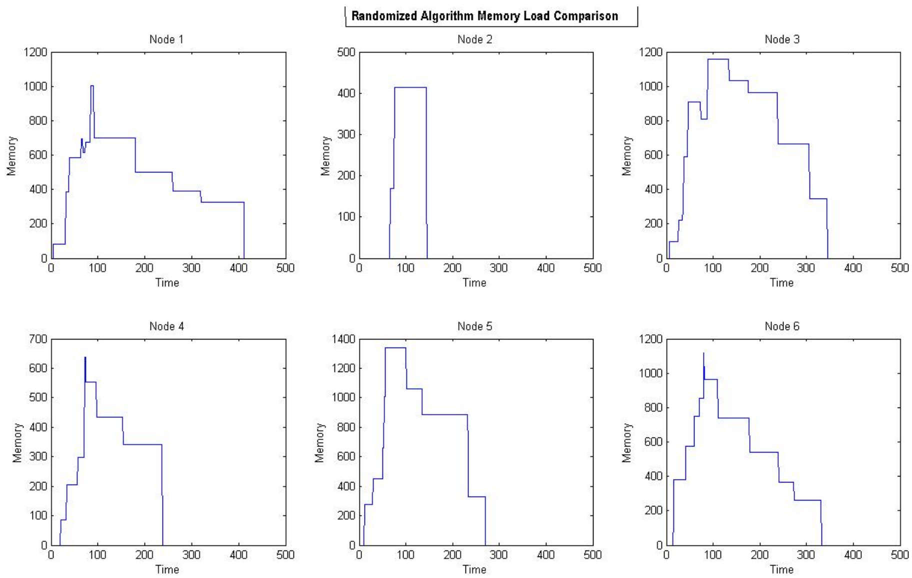

Figure 3.6: Randomized Algorithm Memory Load Profile.

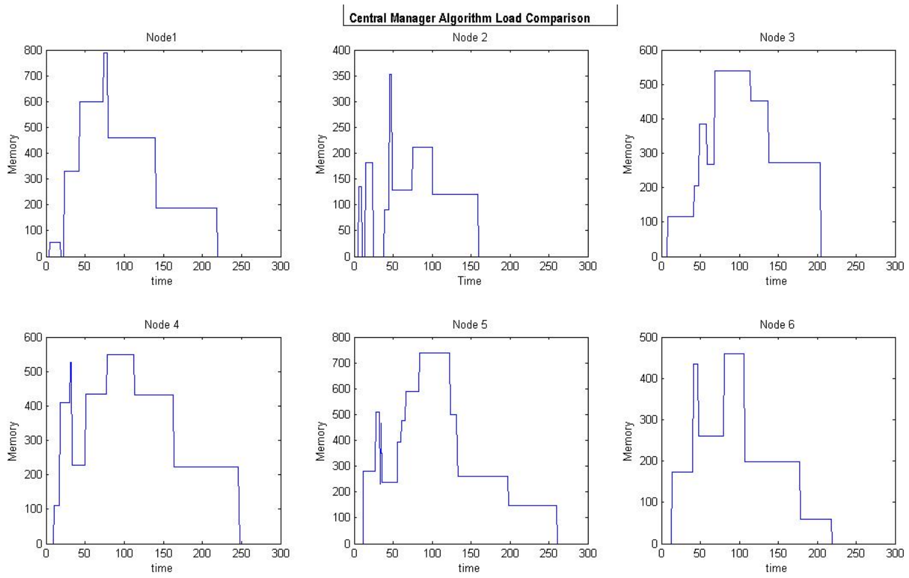

Figure 3.7: Central Manager Algorithm Memory Load Profile.

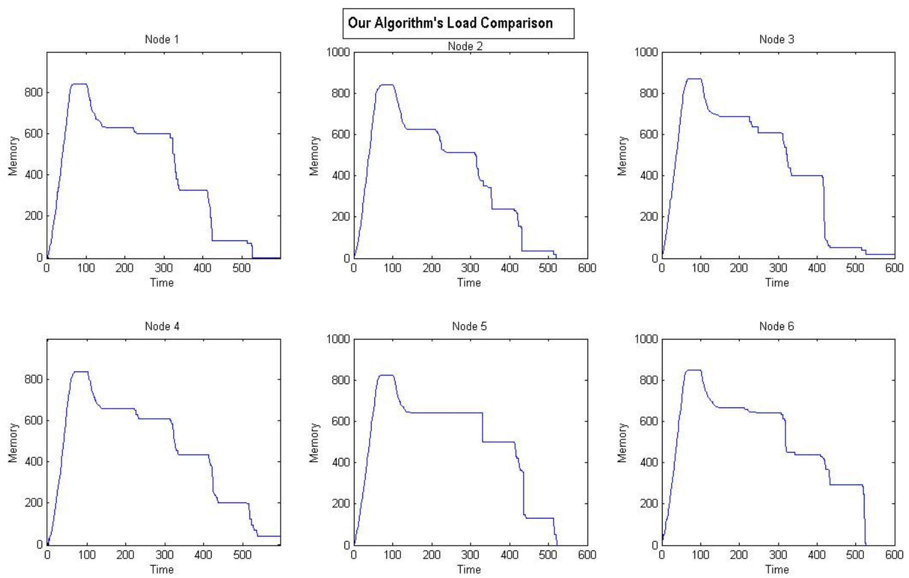

Figure 3.8: Our Load Balancing Algorithm Memory Load Profile.

3.4.2 Simulation Results in Heterogenous Clusters

In Heterogenous clusters the popular WLC algorithm has been simulated along with the proposed algorithm. The results are presented in the following graphs:

Turnaround Times

The proposed algorithm gives results that are worse in Turnaround times to the WLC algorithms:

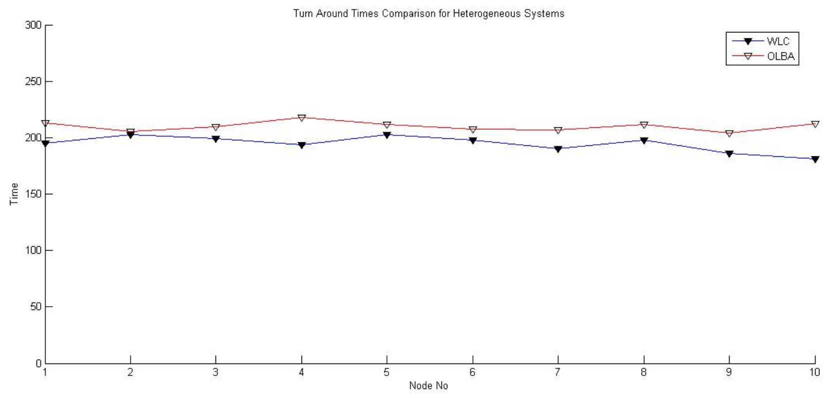

Figure 3.9: Turnaround Times Comparison in Heterogenous Algorithms.

Peak CPU Load

The proposed algorithm gives results that are better in CPU Usage to the WLC algorithms:

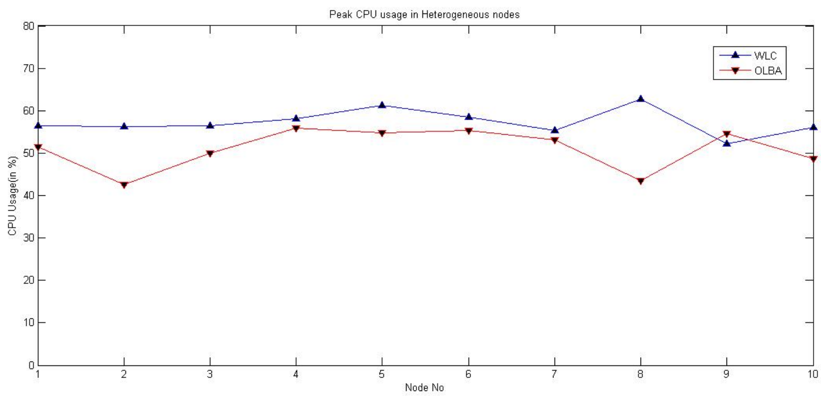

Figure 3.10: Peak CPU Load in Heterogenous Algorithms.

3.5 Summary

The above sections put forth an algorithm to deal with the problem of load balancing in clusters.The algorithm actually works in two phases,of which the first phase is used only if the cluster is heterogeneous.The second phase of the algorithm does the actual load balancing.The algorithm has been simulated and the simulation results show the algorithm to have results comparable or better in most parameters simulated in both homogeneous and heterogeneous clusters.

Chapter 4

Conclusions

Chapter 4

Conclusions

In this thesis we have seen what are clusters, the various properties of clusters and their goals. We have also defined Load Balancing and its goals. We have seen the various types of classifications of Load Balancing Algorithms. We have also studied at least one algorithm of each type. A few algorithms that have been developed on the popular WLC have also been studied. Based on the observations from these algorithms, the thesis then presents a proposal of a Load Balancing Algorithm and it has been backed up with due simulation results and the algorithm shows improvement in several parameters over its peers.This thesis also presents an algorithm that can automate the process of assigning weights to the nodes depending upon their computing power,therby enabling homogenous algorithms to be utilized in heterogenous clusters as well.

Future Work

The future scope of the project is that it can be extended for further number of load unbalancing parameters.It can also be extended to adapt itself to the varing types of load on its own. A performance evaluation of the algorithm with different characteristics of load can also be done.

References

[1] R. Buyya, *High Performance Cluster Computing: Architectures and Systems, Volume 1.* Prentice Hall PTR, 1999.

[2] "`http://en.wikipadea.com`."

[3] D. Bader and R. Pennington, "Cluster computing: Applications," Georgia Tech College of Computing.

[4] L.Wenzheng and S.Hongyan, "A novel algorithm for load bal ancing in cluster systems," in *14th International Conference on Computer Supported Cooperative Work in Design*, 2010.

[5] S.Sharma, S.Singh, and M.Sharma, "Performance analysis of load balancing algorithms," in *World Academy of Science, Engineering and Technology*, 2008.

[6] Z. Xu and R. Huang, "Performance study of load balancing algorithms in distributed web server systems," in *CS213 Parallel and Distributed Processing Project Report*.

[7] H. Stone, "Critical load factors in two-processor distributed systems," in *IEEE Trans. Software Eng., vol. 4, no. 3*, IEEE.

[8] R. Motwani and P. Raghavan, "Randomized algorithms," in *ACM Computing Surveys (CSUR)*, ACM.

[9] L.Rudolph, M.Slivkin-Allalouf, and E.Upfal, "A simple load balancing scheme for task allocation in parallel machines.," in *Proceedings of the 3rd ACM Symposium on Parallel Algorithms and Architectures*, 1991.

[10] W. Leinberger, G. Karypis, and V. Kumar, “Load balancing across near-homogeneous multi-resource servers.,” 2000.

[11] M.Beltran, A.Guzman, and J.L.Bosque, “Dealing with heterogeneity in clusters,” in *Proceedings of the Fifth International Symposium on Parallel and Distributed Computing*, ISPDC.

[12] Y.Wang and R. Morris, “Load balancing in distributed systems,” in *IEEE Trans. Computing.C-34, no. 3*, IEEE.

[13] C.Youn and I.Chung, “An efficient load balancing algorithm for cluster system,” in *H. Jin, D. Reed, and W. Jiang (Eds.): NPC 2005, LNCS 3779*, IFIP International Federation for Information Processing.

[14] P.Werstein, H.Situ, and Z.Huang, “Load balancing in a cluster com puter,” in *Proceedings of the Seventh International Conference on Parallel and Distributed Computing,Applications and Technologies*, IEEE.

[15] P.S.Pacheco, *Parallel Programming with MPI.* Morgan Kaufmann Publishers,Inc.

[16] P.Mohammadpour, M.Sharif, and A.Paikan, “A self training algorithm for load balancing,” in *Fourth International Conference on Networked Computing and Advanced Information Management*, 2008.